Kai Herrmann, Hannes Voigt, Wolfgang Lehner

Online horizontal partitioning of heterogeneous data

Erstveröffentlichung in / First published in:

Information Technology. 2014, 56 (1), S. 4 – 12 {Zugriff am: 05.11.2020}. De Gruyter. ISSN 2196-7032

DOI: https://doi.org/10.1515/itit-2014-1015

Special Issue

Kai Herrmann*, Hannes Voigt, and Wolfgang Lehner

Online horizontal partitioning of heterogeneous data

Abstract: In an increasing number of use cases, databases face the challenge of managing heterogeneous data. Heterogeneous data is characterized by a quickly evolving variety of entities without a common set of attributes. These entities do not show enough regularity to be captured in a traditional database schema. A common solution is to centralize the diverse entities in a universal table. Usually, this leads to a very sparse table. Although today's techniques allow efficient storage of sparse universal tables, query efficiency is still a problem. Queries that address only a subset of attributes have to read the whole universal table including many irrelevant entities. A solution is to use a partitioning of the table, which allows pruning partitions of irrelevant entities before they are touched. Creating and maintaining such a partitioning manually is very laborious or even infeasible, due to the enormous complexity. Thus an autonomous solution is desirable.
In this article, we define the Online Partitioning Problem for heterogeneous data. We sketch how an optimal solution for this problem can be determined based on hypergraph partitioning. Although it leads to the optimal partitioning, the hypergraph approach is inappropriate for an implementation in a database system. We present Cinderella, an autonomous online algorithm for horizontal partitioning of heterogeneous entities in universal tables. Cinderella is designed to keep its overhead low by operating online; it incrementally assigns entities to partition while they are touched anyway during modifications. This enables a reasonable physical database design at runtime instead of static modeling.

Keywords: ACM CCS → Information systems → Database design and models → Physical data models, ACM CCS → Information systems → Database design and models, ACM CCS → Information systems → Database administration → Autonomous database administration.

***Corresponding Author: Kai Herrmann,** Technische Universität Dresden, Germany, e-mail: Kai.Herrmann@tu-dresden.de
Hannes Voigt, Wolfgang Lehner: Technische Universität Dresden, Germany

1 Introduction

Database systems are ubiquitous technological means for managing large amounts of data. As such, databases are exposed to the persistent acceleration of society and technological development. Traditionally modeled database schemas become quickly outdated by reality and application requirements. Where databases should comprehensively capture a large and quickly evolving variety of entities, the traditional database design principles are stretched to their limits. Examples for such application areas are product catalogs (e.g., for electronic devices), clinical findings in patient databases, multi-tenancy databases, or analytic databases, which are constantly enhanced with derived data. Here, frequently new entities appear with new combination of attributes or totally new attributes. Typically, entities show some regularity but not enough to allow modeling a decent database schema.

A common solution in such areas is to model on a more abstract level and centralize the diverse entities in a universal table. Instead of a table for each type of product in a traditional product catalog, the universal table approach has a single product table containing all product properties. Usually, this leads to a very sparse table, which most today's database systems can store efficiently [1–3]. For instance column stores use compression to store universal tables efficiently. Retrieval operations, however, suffer from the universal table modeling. Queries that address only a subset of attributes have to read over the whole universal table including many irrelevant entities that do not have the addressed attributes. Horizontal partitioning presents a simple technique to increase the efficiency of such queries. A partitioning scheme taking into account the irregularity of the entities allows queries pruning partitions of irrelevant entities before they touch the data. Designing and maintaining such a partitioning, though, is a laborious and never ending task most DBAs are not willing to commit to.

In this article, we define the Online Partitioning Problem for heterogeneous data. We sketch the optimal solution to this problem based on hypergraph partitioning and argue that hypergraph approach is inappropriate for an implementation in a database system. Here, a solution with little overhead is a necessity. As the main contribution of the article, we present Cinderella, an autonomous online algorithm for horizontal partitioning of heterogeneous entities in universal tables. Cinderella partitions entities into homogeneous fix-sized partitions, such that the entities in a partition share most of their attributes. Cinderella is designed to keep its overhead low by operating online; it incrementally assigns entities to partition while they are touched anyway during modifications. This enables a reasonable physical database design at runtime instead of static modeling. Queries that retrieve only entities with a subset of attributes can easily prune partitions which contain entities with only irrelevant attributes. This increases the locality of queries and reduces query execution costs.

In the remainder of the article, we define the concrete problem in Section 2 and discuss the optimal solution based on hypergraph partitioning in Section 3. Afterwards, Sections 4 and 5 present Cinderella followed by an evaluation in Section 6. Finally, Sections 7 and 8 briefly discuss related work and conclude the article, respectively.

2 Online partitioning of universal tables

In a universal table, heterogeneous entities are like Swiss cheese. Also known as wide tables, they typically feature a large number of columns while most of the tuples in the table instantiate only a small number of these columns. Figure 1 illustrates such a table for the product catalog scenario. As can be seen, some attributes are very common among the entities, e.g. *name* or *weight*, while others are specific to only certain kinds of entities, e.g. *network* for cell phone. Although the entities exhibit some regularity, it is hard to find reasonable generalizations that allow a stable partitioning scheme regarding the attributes the entities instantiate. While all of today's cameras feature a sensor and screen, some of them are also equipped with GPS sensor and Wi-Fi. In the near future, we may see cameras that may exhibit a mobile connection but lack a storage card slot.

The overall goal of partitioning a universal table is to increase the query efficiency. Each partition is described in the system catalog using a partition synopsis p, which lists the attributes instantiated by the entities in the partition. Likewise, we can list all attributes relevant for a query in a query synopsis q. Queries simply return all entities containing the requested set of attributes. Based on the synopses, queries can easily prune partitions that contain only entities with attributes irrelevant to the query, i.e., partition for which $|p \wedge q| = 0$ holds. Correspondingly, the efficiency of a given partitioning is the ratio of how many entities are relevant for a workload and how many entities are actually read.

Definition 1 (**Partitioning Efficiency**). Given a universal table T containing the entities $\{e_1, e_2, \ldots\}$, a query set $W = \{q_1, q_2, \ldots\}$, and a partitioning $P = \{p_1, p_2, \ldots\}$, the efficiency of P is

$$\textsc{Efcy}(P) = \frac{\sum_{q\in W, e\in T} \operatorname{sgn}(|e \wedge q|) \cdot \textsc{Size}(e)}{\sum_{q\in W, p\in P} \operatorname{sgn}(|p \wedge q|) \cdot \textsc{Size}(p)} \,.$$

The function $\textsc{Size}()$ yields the size of an entity or a partition, indicating how much has to be read to scan the entity or all entities in a partition, respectively. Function $\operatorname{sgn}(arg)$ returns one if its argument arg is positive and zero if arg equals zero to indicate whether the specific entity or partition is accesses.

The aim of online partitioning of a universal table is to continuously maximize the efficiency of a given partitioning under the presence of modification operations. Modification operations are inserts, updates, delete that change the set of entities or vary the set of used attributes.

Definition 2 (**Online Partitioning Problem**). Given a universal table T, a query set W, a partitioning P, and a modification m, online partitioning updates P so that $\textsc{Efcy}(P)$ is maximized for W after m is applied to T.

Relation *Product*

name	sensor res	screen size	network	storage	tuner	rotation	weight	…
Sony SLT-A99	24MP	3in	—	—	—	—	733g	…
Apple iPhone 5	5MP	4in	LTE	32GB	—	—	112g	…
Samsung UE40F6500	—	40in	—	—	DVB-T/C/S	—	9800g	…
WD4000FYYZ	—	—	—	4TB	—	7200rpm	—	…

Figure 1: Example of a Universal Product Relation for Electronic Devices.

The online partitioning problem applies to many different database architectures and to various levels in an architecture. Most obviously in distributed databases or distributed file systems, partitions are distributed among the nodes. In modern main-memory database systems running on a large shared-memory NUMA system, partitions resemble the local memory of each CPU core. In traditional disk-based systems, pages may represent a partition granularity where solving the online partitioning problem can help to increase the query efficiency on universal tables.

3 Optimal solution

For an optimal solution of an instance of the online partitioning problem, we utilize the k-way hypergraph partitioning algorithm [4]. Given an instance of the problem, we can construct a hypergraph such that the partitioning of the nodes using the *connectivity* criterion yields the optimal solution for that instance.

Figure 2 illustrates the optimal solution by a simple example. Given are entities $e_1, \ldots, e_6$ and queries $a, \ldots, f$. For each query only a subset of entities is relevant, e.g., for query e only entity e_4 and e_6 are relevant. In the hypergraph, each node represents an entity and each hyperedge represents a query, such that the edge connects exactly those nodes that are relevant for the query. By k-way partitioning the nodes of the hypergraph, we partition the entities into $k = \lceil \frac{\textsc{Size}(E)\cdot|T|}{\textsc{Size}(P)} \rceil$ partitions. We assume that all entities in T are equally sized and compactly distributed on uniform partitions. The connectivity criterion minimizes the total number of partitions all hyperedges connect. In the context of entity partitioning, the connectivity corresponds to the number of distinct partitions that can contain relevant entities for a query. In the example of

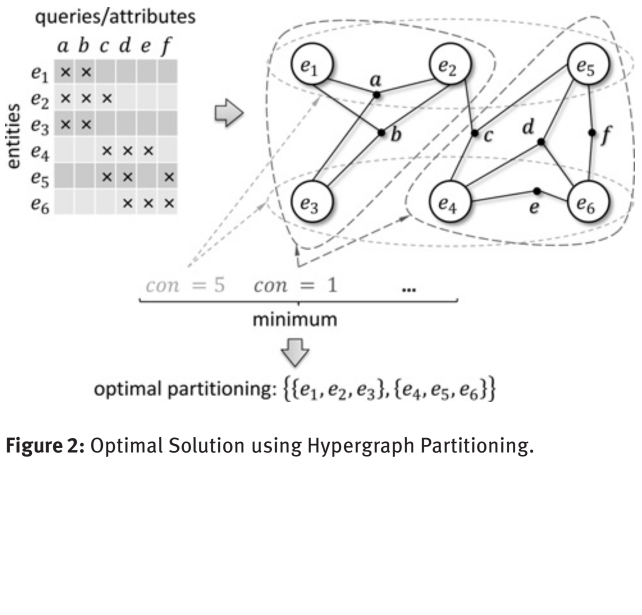
queries/attributes
a b c d e f
entities
e_1 e_2 e_3 e_4 e_5 e_6
a b c d e f
con = 5
con = 1
...
minimum
optimal partitioning: $\{\{e_1, e_2, e_3\}, \{e_4, e_5, e_6\}\}$

Figure 2: Optimal Solution using Hypergraph Partitioning.

Figure 2, the optimal partitioning of the entities $e_1, \ldots, e_6$ for given queries is $\{\{e_1, e_2, e_3\}, \{e_4, e_5, e_6\}\}$. The other partitioning shown in the figure has a high connectivity and therefore is not an optimal solution.

The construction of the graph requires the workload, hence, the result of the partitioning will be tailored for this workload. We call it the *dependent* solution of the problem. Whenever a workload is not available or where the solution should be more general and robust, we can base the hypergraph on the attributes of the entities. The procedure is similar to the one shown in Figure 2, except that $a, \ldots, f$ are attributes, i.e., entity 4 and 6 instantiate attribute e. We call the partitioning scheme resulting from this attribute-based hypergraph the *independent* solution of the problem, as it is independent from a particular workload.

To solve the online partitioning problem, either dependently or independently, we would create the hypergraph for all entities including the one affected by the modification and determine the partitioning. Unfortunately, this is infeasible in practical scenarios as hypergraph partitioning is NP-hard [4]. Even with heuristic hypergraph partitioning algorithms, the solution comes with considerable overhead for two reasons. First, the approach represents each individual entity in the hypergraph, which causes an enormous amount of data accesses. Second, it recalculates the complete partitioning with each modification operations although most of the entities remain unchanged. A practical solution of the online partitioning problem has to exhibit small overhead, such that the overhead does not exceeds the benefit achieved with the partitioning scheme.

4 Cinderella

Cinderella works incrementally. It relies on the basic assumption that the data is already well partitioned. Triggered by a modification operation, Cinderella merely adjusts the partitioning so that the modified entity fits well in. Cinderella creates partitions of a fixed maximum size. Partitions that reach their capacity limit are reorganized with a split operation. We will focus the discussion of Cinderella on the insert operation, since the remaining data manipulation operations are based on the insert. Similar to the optimal solution, Cinderella can create a workload dependent or a workload independent solution. For simplicity in the discussion, we will assume the workload independent setup.

The basic insert procedure is illustrated in Figure 3. Given two partitions cataloged with their synopses and

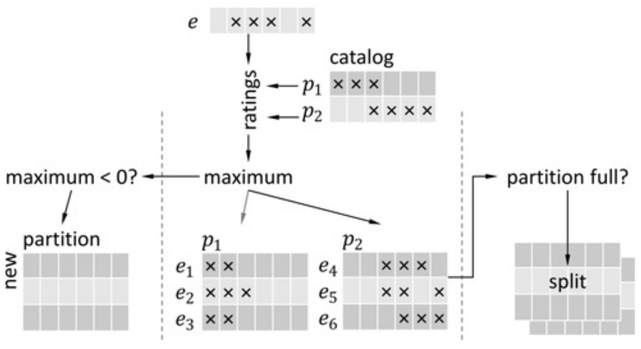

Figure 3: Insert Procedure.

a new entity, Cinderella scans the partition catalog to find the partition which fits best to the entity. Every partition is rated and the entity is inserted to the partition with the highest rating. We will discuss the rating in detail in Section 5. There are two possible exceptions from this basic procedure, illustrated in the left and in the right of the figure. First, the rating can become negative indicating that the new entity fits none of the existing partitions well. In this case, Cinderella creates a new partition for the new entity. Second, the highest rated partition has reached the maximum capacity B so that the entity does not fit in space-wise. Here, Cinderella splits the partition into two new partitions.

The split subroutine consists of two steps. First, Cinderella picks two entities as seeds for the two new partitions. These two seed entities are chosen to be as different as possible regarding the partitioning goal by maximizing the XOR cardinality of the pair's synopses. Second, Cinderella distributes the remaining entities one by one among the two new partitions. It rates all entities against the two new partitions, resulting in two ratings per entity. Cinderella determines the entity–partition pair with the highest rating and inserts the entity into that partition. Cinderella repeats the procedure until no entity is left and the old partition is completely split.

Algorithm 1 lists the complete insert routine. As can be seen the procedure's complexity depends on the number of partitions and the cardinality of the synopses, which is either the number of queries in the workload or the total number of attributes in the universal table. In case of a split, complexity additionally depends on the number of entities in the split partition, which is given by the maximum partition capacity in the system.

The adjustment routines that Cinderella performs for the other modification operations rely on the insert routine. Upon deletes, Cinderella merely removes the deleted entity from its partition. The partitioning itself remains unchanged, except the free space of the affected partition exceeds a configured limit. In this case, Cinderella distributes the remaining entities of the almost empty partition to the other partitions using the insert routine and deletes the partition. Upon updates, Cinderella also runs the insert routine but without actually inserting. In case, the updated entity is assigned to a new partition it is moved. Otherwise, Cinderella updates the entity in place.

Algorithm 1: Online Horizontal Partitioning

```
input: Entity $E$ with Synopsis $S_E$
$bestPartition$, $rating \leftarrow$ rate($S_E$, synopses, $w_p$, $w_s$);
if $rating \geq 0$ then
    if $bestPartition.isFull()$ then
        $P1$, $P2 \leftarrow$ createTwoNewPartitions();
        $entities \leftarrow bestPartition \cup \{E\}$;
        addTwoMostDifferent($entities$, $P1$, $P2$);
        repeat
            $bestEntity'$, $bestPartition'$;
            $bestRating' \leftarrow -\infty$;
            foreach $Entity\ E'$ in $entities$ do
                $P$, $r \leftarrow$ rate($S_{E'}$, $\{S_{P1}, S_{P2}\}$, $w'_p$, $w'_s$);
                if $r > bestRating'$ then
                    $bestRating' \leftarrow r$;
                    $bestPartition' \leftarrow P$;
                    $bestEntity' \leftarrow E'$;
            add($bestEntity'$, $bestPartition'$);
            remove($bestEntity'$, $entities$);
        until $entities.isEmpty()$;
        delete($bestPartition$);
    else
        add($E$, $bestPartition$);
else
    $P \leftarrow$ createNewPartition();
    add($E$, $P$);
```

Assigning entities to partitions during insert and split relies on rating how well an entity and a partition fit together. The next section will detail this rating.

5 Cinderella rating

The Cinderella rating compares an entity synopsis with a partition synopsis to determine how well the entity would fit in the partition. Figure 4 illustrates the comparison. Considering a single attribute, there are four cases: (1) the attribute is in both, entity and partition ($e \wedge p$), which is homogeneity between entity and partition; (2) the attribute is only in the partition ($\neg e \wedge p$), which is a heterogeneity on side of the entity; (2) the attribute is only in the entity ($e \wedge \neg p$), which is a heterogeneity on side of the partition; (4) the attribute is in none of the two ($\neg e \wedge \neg p$). This last case is obviously irrelevant for the rating and is not

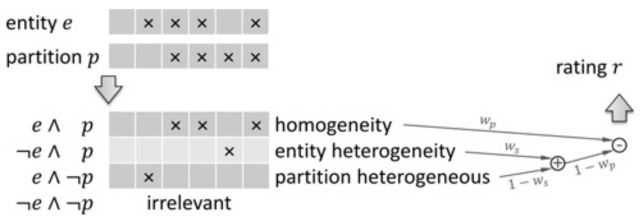

Figure 4: Cinderella Rating.

considered. For the three relevant cases, Cinderella counts the number of matching attributes. The three scores are normalized with the total number of involved attributes $|e \vee p|$.

Homogeneity is the primary score in the rating. It is positive evidence that the entity fits well into the partition and increases the rating. The two heterogeneity scores are secondary in the rating. They represent negative evidence and decrease the rating. However, there is a crucial difference between the two. Partition heterogeneity has a larger impact on homogeneity of a partition if the new entity is added. While heterogeneity on the side of the entity results only in a single entity with fewer attributes than the partition, heterogeneity on side of the partition results in multiple entities with fewer attributes than the partition. Heterogeneity on side of the partition requires updating the partition synopsis, which renders all the old entities in the partition heterogeneous to the partition. To account for these different impacts, Cinderella multiplies the partition heterogeneity score with the size of partition relative to the maximum partition capacity B and the entity heterogeneity score with the relative size of the entity.

Homogeneity score: $h^+ = \frac{|e \wedge p|}{|e \vee p|}$

Entity heterogeneity score: $h_e^- = \frac{\textsc{Size}(e) \cdot |\neg e \wedge p|}{B \cdot |e \vee p|}$

Partition heterogeneity score: $h_p^- = \frac{\textsc{Size}(p) \cdot |e \wedge \neg p|}{B \cdot |e \vee p|}$

Finally, the Cinderella rating is a linear combination of the three normalized scores. For more compact configuration, Cinderella rating expresses three factors in the linear combination with two weights. The primary weight w_p defines the ratio between homogeneity score and the heterogeneity scores. The secondary weight w_s defines the ratio between the two heterogeneity scores. The Cinderella rating calculates as:

$$r = w_p h^+ - \left(1 - w_p\right)\left(w_s h_e^- + (1 - w_s) h_p^-\right)$$

As both the default insert routine and the split operation require the rating, the split uses alternative weights w_p' and w_s'.

6 Evaluation

We implemented a prototype to evaluate Cinderella's partition efficiency, parameter impact, and runtime behavior for data sets of varying irregularity and different partition capacities. To conduct such experiments, we generated synthetic data sets and a corresponding workload, which allowed us to precisely control the irregularity. We preferred synthetic data over real world examples, since there are no known data sets with configurable irregularity. Though, the experiments show that it is important to understand the influence of the irregularity on Cinderella. In the experiments, we inserted the entities of the data set one after another using Cinderella in its workload independent mode. In Section 6.1, we detail the data and workload generation. Sections 6.2 to 6.4 present the results.

Obviously, it is not possible to partition a highly irregular data set perfectly to reach 100% partitioning efficiency. Consequently, the evaluation will include the lower and upper baseline to determine the opportunities. The lower baseline equals the random distribution of entities on partitions. For the upper baseline, we used dependent hypergraph partitioning in the implementation of the framework *patoh* [4]. Due to the complexity, it does not provide the optimum, but a close approximation based on up-to-date algorithms.

6.1 Data set & workload generation

Generating data sets with specific irregularity requires the ability to evaluate the irregularity of a given data set in the first place. Irregularity expresses how far a data set is from fitting into a regular modeled schema. A data set containing groups of entities with perfectly distinct attribute sets is regular. In an irregular data set, entity groups have overlapping attribute sets. Hence, the share of possible differ-

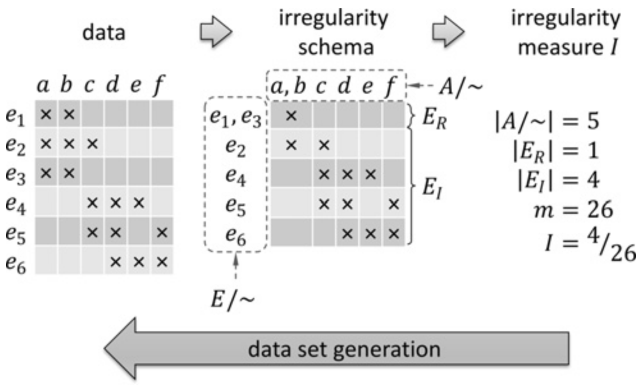

Figure 5: Calculation of Irregularity & Data Set Generation.

ent overlappings of data set is a good measure for the irregularity of the data set.

Figure 5 shows how this irregularity measure is defined. For a given data set, the irregularity schema forms equivalence classes of entities $E/\sim$ and attributes $A/\sim$, so that two entities are equal if they have the same attribute set and two attributes are equal if they are instantiated by the same entities. The entity classes separate into regular ones E_R, whose entities have exactly one attribute classes $A/\sim$ as attribute set, and the irregular ones E_I, whose attribute sets are not a single attribute class. Correspondingly, the total number of possible irregular entity classes m equals the number of possible attribute sets which are not an attribute classes or empty, so that $m = |\mathcal{P}(A/\sim) \setminus (A/\sim \cup \{\emptyset\})|$. Finally, the irregularity I is the ratio between the number of irregular entity classes E_I actually existing in the data set and the total number of possible irregular classes m, so that $I = |E_I| m^{-1}$.

Based on this measure, we generated data sets with specific irregularities by inverting the three steps of Figure 5. First, we fixed the number of attribute classes. In our experiments, we set $A/\sim = 15$, which results in $m = 32\,752$. Second, we filled the irregularity schema with the 15 regular entity classes to ensure that the number of attribute classes is fixed. Subsequently, we added further entity classes to reach the required m. Third, we generated 25 000 entities by randomly picking an entity class for each entity. After generating the data set, we created corresponding queries. The assumption is that the existing entity classes represent common attribute pattern, which are also often accessed by queries. We generated 500 queries, each containing a slightly varied attribute set of a randomly chosen entity class.

6.2 Efficiency

The experiment shown in Figure 6, evaluates the efficiency and the number of splits depending on the irregularity of the data set for a multitude of weightings. This allows determining the average efficiency and the maximum efficiency reached by any weighting. The upper baseline and the lower baseline converge with growing irregularity, as it gets harder to partition an increasingly irregular data set into homogeneous partitions. Cinderella is subjected to the same behavior. The maximum efficiency almost reaches the upper baseline for rather regular data sets. On the other side, it falls to the lower baseline for more irregular scenarios. In these cases the given knowledge and computation time is not enough to reach efficiency improvements online. However, a strong point of Cinderella is, that

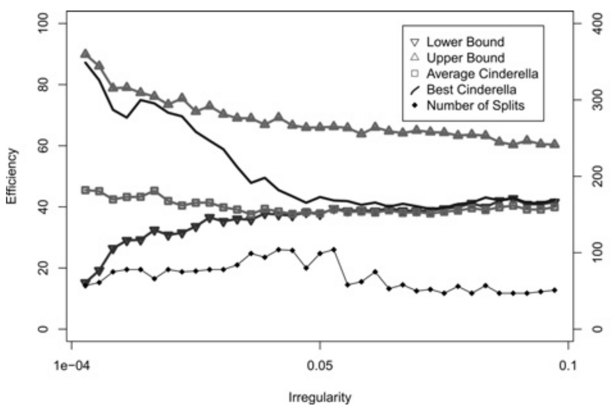

Figure 6: Efficency depending on Iregularity ($B = 50$).

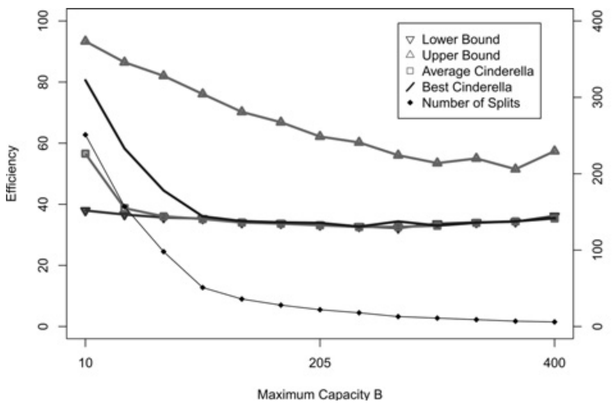

Figure 7: Efficiency depending on Capacity ($I = 0.005$).

it never falls below the lower baseline, which can happen when partitions are not filled. So, Cinderella uses a lot of its potential and provides significant improvements. Furthermore, the number of splits is mostly independent of the irregularity.

Figure 7 shows an experiment, which measures the efficiency of Cinderella's partitionings for varying partition capacities. Again, the upper baseline converges against the lower, as higher capacities cause significantly more accesses for heterogeneous partitions. Cinderella shows similar characteristics. On the one side, it is able to find efficient partitionings for small capacities. On the other side, growing capacities cause Cinderella to fall on the lower baseline. Still, it does not produce worse partitionings than the lower baseline, which is an important achievement. However, we should not forget that larger partitions are in general easier to maintain compared to many small partitions. The number of splits decreases with growing partition capacity, as there are more entities in fewer partitions.

6.3 Parameter impact

We also evaluate the effect of the weights w_p and w_s depending on the scenario. Possible scenarios differ in the irregularity I of the data set and the chosen partition capacity B. Figure 8 shows eight sample scenarios and the achieved efficiency in a heat map where the white parts represent the highest efficiency. This leads to the following four basic conclusions. *Match* factor. On the other hand, the lower the secondary weight w_s the more important the extension of masks compared to the extension of entities.

- In general, w_s is rather low to avoid the extension of synopses and protect the quality. Furthermore, w_p is very robust, but tends to be optimal at 0.5, which shows that especially the *homogeneity* and the *heterogeneity of partitions* are relevant.
- The more irregular the data set is, the narrower the optimal configuration of w_p. As the partitioning task gets harder, it requires more precise decisions.
- The higher the partition capacity, the lower the potential improvement as already shown in Figure 6. Accordingly, the optimal weighting gets more specific for even smaller irregularities. In Figure 8, for instance the measurement with $B = 10$ and $I = 0.06$ is similar to the scenario with $B = 100$ and $I = 0.01$, which shows that the optimal weighting is simply shifted with increasing B.
- When the actual improvement equals zero, the optimal weighting inverts as illustrated in the two most irregular scenarios of $B = 100$. Here, it is dangerous to focus on the heterogeneity of partitions, as entities will chose partitions with full synopses, which is usually caused by a high fill level. This contradicts the idea of preferring emptier partitions to avoid splits. Consequently, the algorithm performs significantly more splits and allocates more partitions than necessary. Unfortunately, the resulting efficiency is lower than the lower baseline, as partitions are not used entirely.

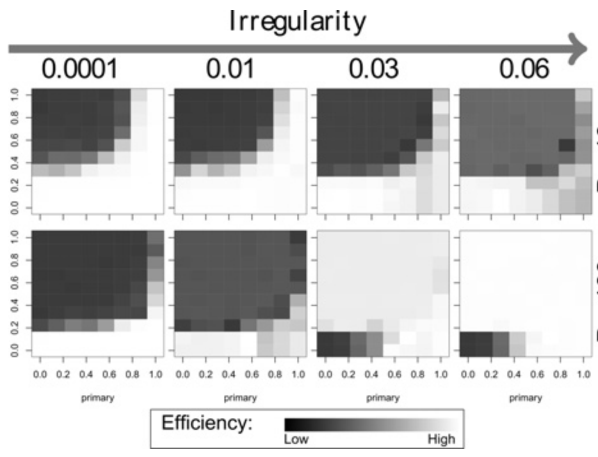

Figure 8: Influence of Weighting on Efficiency.

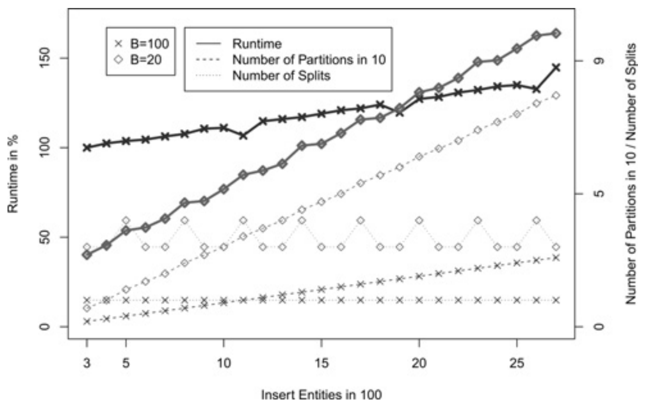

Figure 9: Runtime ($I = 0.005, w_p = w'_p = 0.5, w_s = w'_s = 0.2$).

6.4 Runtime behavior

Finally, we examine the required time of a single run with a fixed irregularity, fixed weighting, and two different partition capacities. After each 100 inserted entities, we measure the required time and the number of splits for the last 100 entities as well as the total number of partitions. We excluded the first 200 entities from the evaluation, as the first inserts often create new partitions instead of splitting existing partitions. After that initial period, Cinderella basically knows the irregularity of the data set and does not receive completely new entities any more. The results in Figure 9 show two linear runtimes, which differ in their base level and the actual increase. The base level of the higher partition capacity is higher even though it executes fewer splits. Nevertheless, splitting larger partitions is even more expensive compared to many small splits, according to the split algorithm's complexity. Even for a growing number of entities in the system, the number of splits remains constant. Therefore, the increase of the required time exclusively depends on the number of partitions, since for each incoming entity all partitions are rated. As, the total number of partitions is higher for small partition capacities, the runtime also increases faster compared to scenarios with larger partitions.

Another interesting aspect of this measurement would be a comparison of different irregularities. As shown in the efficiency measurement, Figure 6, the number of splits is independent of the irregularity, which causes almost equal runtime behavior for different irregularities. To maintain a comprehensible plot, Figure 9 excludes such alternatives.

7 Related work

Handling heterogeneous data sets is no new challenge and has been addressed in multiple research projects. For instance, Wu and Madden build partial indexes on data sets which grow very fast, however, their workloads access mainly small local hot spots. Their idea is to use horizontal partitioning to realize flexible indexing depending on the workload of each partition. The decisions are based on a cost model to simulate potential benefits and risks [5]. In contrast to this, we define heterogeneity on another level and intent to optimize accesses to especially sparse data sets. Another example for handling heterogeneous data sets is proposed by Chu, who uses vertical partitioning to obtain homogeneous parts based on attribute clustering [6]. The retrieved attribute groups infer a hidden schema, which was not explicitly modeled but exists implicitly within the data set. This is also known as Schema Mining.

Schema Mining can be solved by different approaches, like Nestorov who uses fixed point semantics on a graph structure [7]. Furthermore, Bunemann applies graph schemas, which represent the information in a rooted graph with labeled edges. As they have a natural ordering, a schema is inferred by finding the least upper bound of these graphs. Queries, which have to access the derived graph structure, can prune subgraphs to gain a higher performance [8]. Another approach is to use formal concept analysis to extract semantic schema information from a flexible data set with irregular concepts. The set of concepts is iteratively consolidated until it reaches an intended size and can be efficiently handled in a database [9]. Although Schema Mining is very similar to our goal, we laid the focus on storage optimization for efficient query processing, which should be computed efficiently in an online solution.

Furthermore, our work aims at autonomously partition the data set online. A very similar goal has been realized in the *AutoStore* by Jindal [10]. This system autonomously adapts to the workload by monitoring it and triggering reorganizations of the partitioned data set. The main difference to our work is that we partition incoming entities immediately instead of subsequent reorganizations of an existing data set.

8 Conclusion

The universal table is a common setup in databases involving a significant share of heterogeneous, hard to model entities. Horizontal partitioning can help to increase the efficiency of queries on such universal tables. Maintaining such a partitioning poses an optimization problem in the field of physical design. We define this as Online Partitioning Problem for heterogeneous data. Hypergraph partitioning represents an optimal but practically insufficient solution to the problem. With Cinderella, we proposed an autonomous online algorithm for horizontal partitioning of heterogeneous entities in universal tables, which is designed for low overhead. In the evaluation, Cinderella showed capable of significantly increasing the query efficiency over random partitioning. The smaller the partition size the better the achieved efficiency. As smaller partitions, however, increase the number of partitions and thereby the overhead of Cinderella, we will continue our research and aim to further improve Cinderella. Particularly, we will look to improve the management of a large number of partition synopses with specialized data structures and include further aspects of physical database design like caching or indexing.

Received June 10, 2013; accepted October 2, 2013.

References

1. Acharya, S., Carlin, P., Galindo-Legaria, C., Kozielczyk, K., Terlecki, P., Zabback, P. (2008). Relational support for flexible schema scenarios. Proceedings of the VLDB Endowment, 1289–1300.
2. Agrawal, R., Somani, A., Xu, Y. (2001). Storage and Querying of E-Commerce Data. In Enterprise Engineering meets Software Engineering (E2mSE) (pp. 149–158). Morgan Kaufmann Publishers Inc.
3. Beckmann, J. L., Halverson, A., Krishnamurthy, R., Naughton, J. F. (2006). Extending RDBMSs To Support Sparse Datasets Using An Interpreted Attribute Storage Format. In ICDE.
4. Uçar, B., Çatalyürek, Ü. V., Aykanat, C. (2010). A Matrix Partitioning Interface to PaToH in MATLAB. Parallel Computing, 36(5–6), 254–272.

5. Wu, E., Madden, S. (2011). Partitioning techniques for fine-grained indexing. In 2011 IEEE 27th International Conference on Data Engineering (pp. 1127–1138).
6. Chu, E., Beckmann, J., Naughton, J. (2007). The case for a wide-table approach to manage sparse relational data sets. In Proceedings of the 2007 ACM SIGMOD (p. 821). New York, USA: ACM Press.
7. Nestorov, S., Abiteboul, S., Motwani, R. (1998). Extracting schema from semistructured data. ACM SIGMOD Record, 295–306.
8. Peter Buneman, S. D. (1997). Adding structure to unstructured data. In Database (1997) (pp. 336–350). Springer.
9. Mühle, H., Voigt, H., Lehner, W. (2010). Ein begriffsbasierter Ansatz zur semantischen Extraktion von Datenbankschemata. In Enterprise Engineering meets Software Engineering (E2mSE).
10. Jindal, A., Dittrich, J. (2011). Relax and Let the Database Do the Partitioning Online. In M. Castellanos, U. Dayal, W. Lehner (Eds.), BIRTE (Vol. 126, pp. 65–80). Springer.

Dipl.-Inf. Kai Herrmann
Technische Universität Dresden, Database Technology Research Group, Nöthnitzer Straße 46, 01187 Germany,
Tel.: +49-351-46337895
Kai.Herrmann@tu-dresden.de

Kai Herrmann is a PhD student at the Database Technology Group at TU Dresden. He received his Computer Science master's degree from the TU Dresden in April 2013. For his thesis he developed a configurable schema mapping layer which allows flexible management of irregular data sets. From 2009 to 2013, he was a student research assistant at the Database Technology Group focusing on flexible data management.

Dipl.-Inf. Hannes Voigt
Technische Universität Dresden, Database Technology Research Group, Nöthnitzer Straße 46, 01187 Germany
Hannes.Voigt@tu-dresden.de

Hannes Voigt received his Master in Computer Science from the TU Dresden in 2008. Since graduation, he pursues his research activities as a research assistant in the Database Technology Group at TU Dresden. In 2010/2011, he was a visiting scientist at SAP Labs, Palo Alto. His research interests are in flexible data management, graph database, and physical design.

Prof. Dr.-Ing. Wolfgang Lehner
Database Technology Group, Faculty of Computer Science, Technische Universität Dresden, 01062 Dresden, Germany, Tel.: +49-351-46338383
Wolfgang.Lehner@tu-dresden.de

Wolfgang Lehner received his Master, Ph. D., and habilitation in Computer Science from the University of Erlangen-Nuremberg. Since 2002, Wolfgang Lehner is full professor and head of the Database Technology Group at TU Dresden. He was a visiting scientist at IBM Almaden, Microsoft Research Redmond, and SAP Walldorf. His major research focuses on the efficient processing of empirically collected mass data with advanced database technology.